REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 23-07-2014	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* 06 Mar 2012 – 05 Mar 2014

4. TITLE AND SUBTITLE Adaptive problem solving	**5a. CONTRACT NUMBER** FA2386-12-1-4018
	5b. GRANT NUMBER Grant AOARD-124018
	5c. PROGRAM ELEMENT NUMBER 61102F
6. AUTHOR(S) Prof. Michael Barley	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Auckland 38 Princes Street Auckland 1010 New Zealand	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AOARD UNIT 45002 APO AP 96338-5002	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/AFOSR/IOA(AOARD)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AOARD-124018

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A: Approved for public release. Distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research aims to eventually create problem solvers that automatically adapt their problem solving techniques, representations, and heuristics to fit the current problem and computing environment. In this report, we discuss our prototype, RA*, which automatically adapts its heuristics to the current problem and computing environment. RA* does this by computing the impact of its candidate heuristics upon its ability to solve the problem in this environment and using the heuristic with the most impact. Our experiments show that RA* can solve more of the 2011 IPC Deterministic Optimization Track's problems than current state-of-the-art heuristic selection systems and RA* can solve them faster.

15. SUBJECT TERMS

algorithms, Computer Science

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Brian Sells, Lt Col, USAF
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			
U	U	U	UU	10	**19b. TELEPHONE NUMBER** *(Include area code)* +81-3-5410-4409

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Final Report for AOARD Grant 124018

"Adaptive Problem Solving"

5 June 2014

Name of Principal Investigator: Michael W. Barley
- e-mail address : barley@cs.auckland.ac.nz
- Institution : University of Auckland
- Mailing Address :
  Department of Computer Science
  The University of Auckland
  Private Bag 92019
  Auckland 1142
  New Zealand
- Phone : (64)9-373-7599 x86133
- Fax : (64)9-373-7453

Period of Performance: 2012/03/06 - 2014/03/05

Abstract:

This research aims to eventually create problem solvers that automatically adapt their problem solving techniques, representations, and heuristics to fit the current problem and computing environment. In this report, we discuss our prototype, RA*, which automatically adapts its heuristics to the current problem and computing environment. RA* does this by computing the impact of its candidate heuristics upon its ability to solve the problem in this environment and using the heuristic with the most impact. Our experiments show that RA* can solve more of the 2011 IPC Deterministic Optimization Track's problems than current state-of-the-art heuristic selection systems and RA* can solve them faster.

Introduction:

Ideally, all a user would need to give a general problem solver is a description of the problem and the problem solver would adapt its solution methods, representations, and heuristics to fit its current problem and computing environment[1]. Unfortunately, far too often today's users must adapt their problems to fit the problem solver. This research aims to change that.

Artificial Intelligence (AI) has been very successful in providing a general approach to problem solving. Along with this approach, AI has come up with a large number of solution techniques, problem representation schemes, and types of heuristics. Unfortunately, no one combination is best for all problems. Worst still is that solution techniques, problem representations, and heuristics all interact and we do not yet understand the nature of this interaction sufficiently to efficiently search for the right combination for a given problem and computing environment.

This is the beginning of our research on this challenge. Instead of trying to find the right combination of these components, we focus on a single component, heuristics. Specifically, how to efficiently search for the right heuristic for a given problem, computing environment, solution technique, and problem representation. As we succeed with this, we will move moving on to doing this individually for the other two components (techniques and representation), and after that we will look at the problem of finding the right combination across all three components.

In order to leverage our experience with heuristics to the other components, there are a

[1] The computing environment includes the processor speed, amount of memory, and availability of co-processors.

number of considerations that our approach should satisfy:

- It searches through an extremely large number of candidates.
- It predicts the impact of the different candidates upon the resources needed to find the solution.
- Those predictions are accurate enough that they can be used to choose an appropriate candidate that significantly shortens the search time for the solution.
- Making these predictions should be cheap enough that their cost is not greater than the time saved by using their choice of heuristic.

Theory:

Our problem solver, RA*, is built on top of the Fast Downward planner (Helmert 2006). Fast Downward comes with a number of heuristics and heuristic generators. Standard Fast Downward allows the user to select which heuristic/heuristic generator is to be used with its version of A*. However, RA* uses Fast Downward's heuristics and heuristic generators to provide a reasonable number of heuristics for a given problem. RA* will then decide which combination of heuristics to use in solving the problem.

Predicting the Best Combination

The core idea of this research is that the problem solver decides on the combination to use by predicting which one will result in the shortest runtime to solve the problem. For a given problem, the problem solver uses the following formula to predict runtimes:

$$t_c(f) \approx ((b_c)^f * p_c(f)) * (tx + tv_c) \qquad \textit{(formula 1)}$$

Where $\boldsymbol{t_c(f)}$ is the time it takes the problem solver to expand all the nodes with an f-value of *f*, using heuristic combination *c*. $\boldsymbol{b_c}$ is the heuristic branching factor when using heuristic combination *c*. $\boldsymbol{p_c(f)}$ is the probability that a node's state is <u>not</u> the same as one that has already been expanded when using heuristic combination *c* to expand nodes with f-values *f*. $\boldsymbol{tv_c}$ is the average time it takes heuristic combination *c* to evaluate a state, and $\boldsymbol{tx}$ is the average time it takes to expand a node without evaluating its state. For a given problem and heuristic combination, *c*, our predicted runtime for solving the problem is simply: $\sum_{i=0}^{d} t_c(i)$, where *d* is the optimal solution path length of the problem. Some of these terms, e.g., $\boldsymbol{tv_c}$ and $\boldsymbol{tx}$, have values that can be measured when the problem solver starts the search process. We call these *a priori* terms. Other terms, e.g., $\boldsymbol{b_c}$ and $\boldsymbol{p_c(f)}$, have values that can be estimated while the problem solver is engaged in solving the problem. We call these *a posteriori* terms. The values of all of these terms are problem specific.

In general, because each combination may have a different branching factor and a different per node cost, no one heuristic combination will be best for all *f-levels.* Heuristic combinations that are cheap but with higher branching factors are normally best at low *f-levels* and heuristics with low branching factors but higher per node costs are normally best at higher *f-levels.* This means that which combination is best depends upon which *f-level* we are interested in. This allows us to use formula 1 in a number of ways. Assuming we have values for our *a priori* and *a posteriori* terms then we can use formula 1 to predict for which *f-levels* one combination will be faster than another combination. If we knew the optimal solution path length for the given problem then we could use the formula to predict which combination will enable the problem solver to find that solution in the least time. If we did not know the optimal solution path length but had a deadline, then we could predict the combination which would get to the highest *f-level* within that time limit and consequentially have the best chance of finding the solution before that deadline. We could also use the formula to just predict which combination would be best for the next *f-level,* which is what RA* uses the formula for.

Computing Values for Non-Branching Factor Terms

In order to use formula 1 for any of the above purposes, we need to have values for our terms. The ***tx*** value is measured on the root node of A*'s search graph. The evaluation time for each individual heuristic is also measured at the root node. When we need the evaluation time, $\boldsymbol{tv_c}$, for a particular combination, *c*, we simply add together the evaluation times for the individual heuristics comprising c. The value for $\boldsymbol{p_c(f)}$ is approximated by the ratio of the number of nodes with f-value of *f* which were evaluated (which means that they were not the same as a previously expanded state) over the number of nodes that were generated with that f-value.

Computing the Values for the Combinations' Branching Factors Terms

Computing the branching factor for a specific combination of heuristics is relatively straightforward as long as we ignore the cost of computing it. Using that combination of heuristics, the search graph is grown to a sufficient f-level that the branching factor is relatively stable, then the number of nodes generated at that f-level is counted and as well as the number of nodes generated at the next f-level. The branching factor for that combination of heuristics is the ratio of the latter count over the former count. While growing a search graph for one combination is feasible, we are talking about large numbers of combinations. Our largest set of heuristics had 120 heuristics, which translates into 2^{120} (or 10^{36}) combinations, this rules out growing a search graph for each combination. What we would like to do is to grow a single search graph which allows the system to compute the branching factor for any of those 2^{120} combinations. We will now briefly describe how RA* does this.

As we saw above, computing the branching factor is counting the evaluated nodes and the generated nodes, and taking the ratio of these two counts. Initially RA*'s branching factor is erratic. However, RA* expands the initial part of the search until the branching factor becomes relatively stable and samples the nodes keeping track of which heuristics are responsible for those nodes' expansions. This rest of this process is described in more detail in (Franco et al., 2013).

Experiments and Analyses:

We would like to know how well our approach works with respect to two reference points. One is with respect to picking the best possible subset for each problem over some representative range of problems. The other is with respect to the current state-of-the-art approaches. This section describes experiments that show our approach enables a problem solver to solve more problems with less search effort and that the actual overhead of our approach[2] can be quite small.

Experimental Details

To do these comparisons we need a performance benchmark. The most commonly used benchmark is performance on the problems and domains of the past International Planning Competitions (IPC), the last one was in 2011 and the next one is in June 2014. Each IPC has a number of different tracks representing different types of planning problem domains and tasks. We have chosen the 2011 IPC's Deterministic Optimization Track as our benchmark. This track has 14 domains and each domain has 20 problems. To make the comparisons as fair as possible the subsets of heuristics are run with exactly the same planner, so that differences in performance can be attributed to differences in the quality of the subset of heuristics chosen. Fast Downward is currently the planner of choice for benchmarking. It is publically available, highly customizable and has a large number of heuristics and heuristic generators already built in. We do all the comparisons we talk about in this report using Fast Downward.

[2] This refers to our approach to selecting a combination of heuristics, not to generating the basic set of heuristics which can be quite expensive.

In the IPCs, systems are ranked by the number of problems they are able to solve within the given memory and time limits. For the 2011 IPC, problem solvers are only given 6 GBs of memory and only allowed 30 cpu minutes per problem. We will follow this ranking system.

While we would like to find out what is the best performance possible from selecting a subset of our 45 available heuristics, the straightforward approach of just running the benchmark on each of the 2^{45} possible subsets is not feasible. Neither is running the benchmark on each of the 20,000 subsets generated by our approach feasible. If we assume that each of the 192 problems take an average of one minute then running the benchmark on each of the subsets would take 7.3 cpu years. We are currently considering branch-and-bound approaches to estimating the best possible performance achievable by any of the 20,000 subsets, this would allow us to assess how close to optimal our selection is on average.

So how does RA* fare against state-of-the-art systems? Unfortunately, it is hard to do a totally fair comparison. There are three approaches currently being used: "generate and select", select on a node-by-node basis, and default selections. Three of the systems that do the first approach (iPDB (Haslum,et al. 2007), GA-D, and GA-ND (Edelkamp 2007)) combine the processes of generating and picking the heuristic so that it's hard to know how well they would do given our set of heuristics. The last system (Rayner et al., 2013), using the first approach, does select a subset out of a given set of heuristics. However, since it selects its subset based on minimizing the size of search tree, it would be unlikely to compete successfully when measured by solution time.

There have been three systems (*selmax* (Domshlak et al. 2012), Lazy A* and Rational Lazy A* (Tolpin et al. 2013)) that take the second approach. However, they only pick from two heuristics not 45. They pick which heuristic to apply to the current node. Their results so far have proved promising, but it's not yet obvious how easily this approach will scale up.

There are two standard default methods for using a set of heuristics. One (MAX) is to use all of them and take their maximum value, the other (RAND) is to pick a heuristic at random at each node and use its value. The former produces the smallest possible search tree for that set of heuristics, but also has the highest cost per node. The latter produces a larger search tree but has a much lower cost per node.

As our state-of-the-art heuristic selection systems, we have chosen iPDB, GA-D, GA-ND, MAX, RAND, and two standard heuristics LM-cut and h^{max}. LM-cut is a very costly but very accurate state-of-the-art heuristic and h^{max} is, like RAND, a very cheap but not as accurate heuristic. We have modified GA-ND and GA-D to return their top 21 heuristics instead of just their top heuristic. The performance we report for GA-ND and GA-D is for the modified versions. Thus we have a set of seven systems to compare against and 45 heuristics for each problem from which to choose a subset.

We have run experiments to answer two questions: (1) How expensive is our approach? and (2) How does its results compare to state-of-the-art systems? The experimental results we review here are reported more extensively in (Barley et al., 2014).

Cost of Our Approach

	Avg Per Problem	Standard Deviation	Ratio
Heuristic Generation	196.51	200.63	69.0%
Sampling/Selection	2.37	15.57	0.8%
Search For Solution	86.09	225.17	30.2%
Total	284.97	441.37	100.0%

Table 1: RA*'s Runtime Breakdown Per Problem

In our experiments, RA*'s average runtime cost per problem, to do the sampling and compute the estimated runtimes of all the heuristic combinations that are likely to be chosen, is 2.37 seconds, the standard deviation is 15.57 seconds. So for a set of 45 base heuristics, the total selection cost is quite a small part (0.8%) of RA*'s average total runtime per problem. The majority of the runtime (69.0%) is spent generating heuristics.

To understand how RA* manages to spend so little on choosing its combination of heuristics, we need to look at what we mean by "all the heuristic combinations that are likely to be chosen". As mentioned earlier, given *n* heuristics, there are 2^n combinations. For large enough *n,* RA* will not, in any reasonable amount of time, be able to estimate the runtimes for all combinations. Fortunately, this is not necessary. There is a diminishing return of adding new heuristics to a combination (Zhavi, et al., 2007). This is where the advantage of adding another heuristic to the combination decreases with the number of heuristics already in the combination. This leads to the situation where adding more heuristics will actually lead to increases in runtime.

While RA* does not check for this condition, the current default settings, for determining the maximum size of combinations, seem to satisfy this condition. While RA* is sampling, it prunes away heuristics deemed to be too weak to be useful. At the end of the sampling period, RA* uses the number of heuristics remaining to determine the maximum number of heuristics it will allow in any combination. By default, RA* looks at no more than 20,000 combinations. RA* determines the maximum combination size that will create less than 20,000 combinations. RA* exhaustively looks at every combination of the remaining heuristics up to that maximum combination size. It is RA*'s pruning of the weak heuristics and subsequent limiting of the number of combinations that it will consider, that enables RA* to spend so little time selecting its combination of heuristics.

Comparing RA's Performance with State-of-the-Art Systems*

RA*'s cost of selecting its combination of heuristics may be low, but how does its choice compare to what current state-of-the-art systems are able to do?

As a sanity test we first check to see how RA* would have fared in the 2011 competition. RA* solved 187 problems and Fast Downward-StoneSoup (Helmert 2011), the 2011 winner, solved 185 problems. Fast Downward-StoneSoup is a portfolio planner and the differences between it and RA* go well beyond just which heuristics they chose. While we could not run under a compute environment identical to the one used for the competition, we believe that our processor was not significantly faster than the ones used in the competition. It seems clear that RA* is competitive with the best of the state-of-the-art problem-solvers.

To see if our approach is an improvement on the state-of-the-art heuristic combination selection systems, we compared RA*'s performance against the seven other systems. Our results come from analyzing the performance of all eight systems on the 192 problems (out of the 280 problems available) that were solved by at least one of the eight systems. Fuller descriptions of the individual systems can be found in (Barley et al., 2014). Table 2 shows the performance of each of the eight systems on the 192 problems. The "Avg" column shows the average time each system took on each of the problems. The "Sum" column shows the total time each system took on all 192 problems. RA*, on average, took only 70% of the time that the next fastest system (GA-ND) took on these problems. In the IPC, each system is limited to 30 cpu minutes per problem and to a six GB memory limit. RA* did not generate any new types of heuristics, it simply chose a subset of the forty-five heuristics already created for the problem.

	Avg.	Std. Dev.	Sum
RA*	88.46	240.74	17,000
GA-ND	123.85	280.17	23,800
GA-D	133.83	314.39	24,900
iPDB	212.01	455.71	40,700
MAX	260.91	500.40	50,100
RAND	321.23	505.60	59,600
LM-cut	446.62	699.15	85,800
h^{max}	706.12	820.01	136,000

Table 2: Search Time Per Problem

If we look at Table 3, we see that although RA* solved more problems than the other systems and spent significantly less time, on average, searching for the solution than the other systems, it did not solve proportionally more problems. Part of the reason for this is undoubtedly because the problems within a domain had a wide range of difficulties (i.e., optimal solution path lengths). For those domains with a high branching factor, a problem with a solution just one step longer could take an order of magnitude longer to solve. For these domains, all the systems could only solve the easy problems and nothing else (e.g., barman). If the branching factor was small enough then all the systems could solve all of the problems (e.g., sokoban). However, for many domains, most of the systems could find the shorter solutions and none could find the longer solutions. For these domains with moderate branching factors, the difference between the systems' ability to solve a problem was the depth on which that transition from solvable to non-solvable[3] occurred.

	Problems Solved RA* problems solved (problems solved after sampling phase)								
	RA*	GA-ND	GA-D	MAX	iPDB	LM-cut	RAND	h^{max}	total
Barman	4(4)	4	4	4	4	4	4	4	4
Elevators	19(9)	19	19	18	16	18	16	13	19
Floortile	6(4)	3	4	6	2	6	2	6	6
Nomystery	20(4)	20	20	20	18	14	20	8	20
Openstack	15(10)	16	16	13	16	16	15	16	16
Parcprinter	13(3)	13	13	13	11	13	10	11	13
Parking	7(7)	1	1	1	7	1	0	0	7
Pegsol	19(14)	19	19	17	18	17	17	17	19
Scanalyzer	14(4)	10	9	11	10	11	6	6	14
Sokoban	20(9)	20	20	20	20	20	20	20	20
Tidybot	13(11)	12	11	11	13	12	7	5	13
Transport	9(4)	10	10	8	6	6	7	6	10
Visitall	18(2)	16	17	17	16	10	13	9	18
Woodworking	10(5)	10	9	11	7	11	5	4	13
Total	187(92)	173	172	170	164	159	143	123	192
Ratio	.97(.49)	.89	.88	.88	.85	.83	.74	.64	

Table 3: Problems Solved for Each System for Each Domain

The 30 minute time limit applies not only to the search process but also to the total time taken by all the processes, including the heuristic generation process. So, another possible explanation for why RA* did not solve proportionally more problems is that while it spent much less time per problem and solved more problems, it also spent a lot more generating its

[3] Non-solvable in the sense that Fast Downward is no longer able to solve it within the IPC constraints.

sets of heuristics than its closest competitors GA-ND and GA-D. On average RA* spends 70% of its time on generating its set of 45 heuristics, while GA-ND spends about 50% of its time on heuristic generation and GA-D only spends 35%. Table 4 shows the average percentage of processing time spent on generating heuristics for each system. Both h^{max} and LM-cut do not produce heuristics (they are themselves heuristics), so their heuristic generation time is zero.

	Avg	Std. Dev.	Sum	Ratio
RA*	196.51	200.63	37,729.59	69.0%
GA-ND	108.84	111.04	20,896.48	46.8%
MAX	192.40	201.25	36,940.87	42.4%
RAND	196.45	208.44	37,700	37.9%
GA-D	71.96	98.90	12,815.52	35.6%
iPDB	15.67	25.24	3,008.4	6.6%
h^{max}	0	0	0	0%
LM-cut	0	0	0	0%

Table 4: Runtime for Generating Heuristics

	Avg.	Std. Dev.	Sum
GA-D	205.79	371.40	38,700
iPDB	227.68	469.94	43,700
GA-ND	232.68	333.67	44,700
RA*	284.97	319.38	54,700
LM-cut	446.62	699.15	85,800
MAX	453.31	571.95	87,000
RAND	517.68	607.15	99,400
h^{max}	706.12	820.01	136,000

Table 5: Average Total Time Per Problem

While RA* solved more problems and solved them quicker than the other systems, RA* also spent more time generating its set of heuristics than any of the non-default systems (since RA* used those non-default systems to generate its heuristics, it actually used the same amount of time as all of them combined). Table 5 shows the impact RA*'s larger heuristic generation time has on its total time compared to the other systems. RA* is 38% slower than the fastest overall system, GA-D, when total time is considered, even though RA* is 51% faster than GA-D when only their search times are considered. This suggests that the weakest part of RA* is its current reliance on other systems to generate its set of basic heuristics.

Results and Discussion:

This research explores whether a problem solver can be tailored to a specific problem and computational environment and produce state-of-the-art performance. We want problem solvers that can dynamically tailor which solution techniques, problem representations, and search heuristics they use to solve a given problem. Unfortunately, the field does not yet sufficiently understand the interactions between these components to address the entire problem. Instead, this research is focusing on dynamically tailoring which heuristics to use with a specific problem and computational environment. We wanted our system to do the following:

- Search through an extremely large number of candidates.
- Predict the impact of the different candidates upon the resources needed to find the

solution.

- Those predictions are accurate enough that they can be used to choose an appropriate candidate that significantly shortens the search time for the solution.
- Make these predictions cheap enough that their cost does not exceed the time saved by using their choice of heuristic.

In our experiments, RA* gathered information on 2^{45} combinations of heuristics and predicted the runtime of up to 20,000 of those combinations. These predictions were good enough to enable RA* to both solve more problems and to solve them quicker than state-of-the-art heuristic selection systems on the IPC 2011 benchmark. While the cost of making our predictions is very cheap (only 0.8% of RA*'s total runtime cost), the cost of using these other systems to generate the set of base heuristics is very expensive (69% of RA*'s total runtime cost). Future research will investigate the possibility of incorporating the process of generating the basic heuristics into this framework. In particular, we will look at the possibility of integrating the generation process into the evaluation procedure.

References

Barley, M., Franco, S. & Riddle P. "Overcoming the Utility Problem in Heuristic Generation: Why Time Matters" in Proceedings of the International Conference on Automated Planning and Scheduling, June 2014.

Domshlak, C., Karpas, E., & Markovitch, S. "Online Speedup Learning for Optimal Planning" in Journal of Artificial Intelligence Research, Volume 44, 709-755, 2012.

Edelkamp, S. "Automated Creation of Pattern Database Search Heuristics" in Model Checking and Artificial Intelligence, Springer Lecture Notes in Computer Science, Volume 4428,35-50, 2007.

Franco, S., Barley, M., & Riddle, P. "A New Efficient In Situ Sampling Model for Heuristic Selection in Optimal Search" in AI 2013: Advances in Artificial Intelligence, Springer Lecture Notes in Computer Science, Volume 8272,178-189, December 2013.

Haslum, P., Botea, A., Helmert, M., Bonet, B. & Koenig, S. "Domain-independent construction of pattern database heuristics for cost-optimal planning" in Proceedings of Association for Advancement of Artificial Intelligence, 2007.

Helmert, M. "The Fast Downward Planning System", in Journal of Artificial Intelligence Research, Volume 26, 191-246, 2006.

Helmert, M., Röger, G. , Seipp, J., Karpas, E., Hoffmann, J., Keyder, E., Nissim, R., Richter, S., & Westphal, M. "Fast downward stone soup." *Seventh International Planning Competition (IPC 2011), Deterministic Part* (2011): 38-45.

Rayner, C., Sturtevant, N., & Bowling, M. "Subset selection of search heuristics" in Proceedings of the International Joint Conference on Artificial Intelligence, 637-643, 2013.

Tolpin, D., Beja, T., Shimony, S., Felner, A., & Karpas, E. "Towards rational deployment of multiple heuristics in A*" in Proceedings of the International Joint Conference on Artificial Intelligence, 674-680, 2013.

Zahavi, U., Felner, A., Schaeffer, J., & Sturtevant, N. "Inconsistent Heuristics" in Proceedings of Association for Advancement of Artificial Intelligence, 2007.

List of Publications and Significant Collaborations that resulted from your AOARD supported project:

Barley, M., Franco, S. & Riddle P. "Overcoming the Utility Problem in Heuristic Generation: Why Time Matters" in Proceedings of the International Conference on Automated Planning and Scheduling, June 2014.

Franco, S., Barley, M., & Riddle, P. "A New Efficient In Situ Sampling Model for Heuristic Selection in Optimal Search" in AI 2013: Advances in Artificial Intelligence, Springer Lecture Notes in Computer Science, Volume 8272,178-189, December 2013.

Franco, S., Barley, M., & Riddle, P. "In Situ Selection of Heuristic Subsets for Randomization in IDA* and A*" in Proceedings of the Heuristics and Search for Domain-Independent Planning Workshop, 5-13, June 2013.

Attachments: Publications a), b) and c) listed above if possible.

DD882: As a separate document, please complete and sign the inventions disclosure form.

Important Note: If the work has been adequately described in refereed publications, submit an abstract as described above and refer the reader to your above List of Publications for details. If a full report needs to be written, then submission of a final report that is very similar to a full length journal article will be sufficient in most cases. This document may be as long or as short as needed to give a fair account of the work performed during the period of performance. There will be variations depending on the scope of the work. As such, there is no length or formatting constraints for the final report. Keep in mind the amount of funding you received relative to the amount of effort you put into the report. For example, do not submit a $300k report for $50k worth of funding; likewise, do not submit a $50k report for $300k worth of funding. Include as many charts and figures as required to explain the work.